John von Neumann Institute for Computing

Solving Real Life Applications With High Accuracy

C.A. Hölbig, P.S. Morandi Jr., D.M. Claudio, T.A. Diverio

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 317-324, 2006.

http://www.fz-juelich.de/nic-series/volume33

Solving Real Life Applications With High Accuracy

Carlos Amaral Hölbig[a], Paulo Sérgio Morandi Júnior[b], Dalcidio Moraes Claudio[c], Tiarajú Asmuz Diverio[b]

[a]Universidade de Passo Fundo – ICEG – Passo Fundo – Brazil

[b]Universidade Federal do Rio Grande do Sul – II and PPGC – Porto Alegre – Brazil

[c]Pontifícia Universidade Católica do Rio Grande do Sul – Porto Alegre – Brazil

1. Introduction

In this paper we discuss the implementation of methods to solve linear systems, with high accuracy, used in real life applications. It is important because many different numerical algorithms of these applications contain the solution of linear system (1) as a subproblem. Because of these aspects, this work aims the development of solvers to linear systems (for dense and sparse matrices) with high accuracy on cluster computers using C–XSC library. The methods implemented are used in real life applications like hydrodynamic, agriculture and power electric systems.

$$Ax = b \tag{1}$$

In our research some programs were developed in C–XSC with the high accuracy characteristic, where the results are obtained with a good quality. The C-XSC library is a (free) C++ class library for scientific computing for the development of numerical algorithms delivering highly accurate and automatically verified results by use of the interval arithmetic (see details about this library in [1] and [2]). It provides a large number of predefined numerical data types and operators. These types are implemented as C++ classes. Thus, C-XSC allows high-level programming of numerical applications in C++. Actually, our software run on clusters at UFRGS (Brazil), UPF (Brazil) and Wuppertal (Germany).

During the development of this work was necessary to available a high performance (cluster computers) and high accuracy (by use of C–XSC library) environment. Because of this, our work was divided in 4 (four) steps:

- Integration between the libraries C–XSC and MPI on clusters (section 2);
- Development of solvers for linear systems (section 3);
- Implementation of basic tests in the environment (section 4);
- Solve real life applications with high accuracy (section 5).

Because of these aspects, this work aims the development of solvers with high accuracy for linear systems of equations and the adaptation of the algorithms implemented to cluster computers using C–XSC library. This library is available for download in *www.math.uni-wuppertal.de/wrswt/index_en.html*. Our solvers work with dense and sparse (in special banded matrices) linear equation systems. Nowadays, the solver for dense matrices works with all four basic numerical C–XSC data types: *real, interval, complex*, and *complex interval* and the solver for sparse matrices works with *real* and *interval* data types. All our programs are freeware [3].

2. Integration between C–XSC and MPI Libraries

As part of our research, we done the integration between C–XSC and MPI libraries on cluster computers. This step was necessary and essential for the adaptation of our solvers to high performance environments. This integration was developed using, initially, algorithms for matrix multiplication in parallel environments of cluster computers. We done some comparations about the time related to the computational gain using parallelization, the parallel program performance depending on the matrix order and the parallel program performance using a larger number of nodes. We also studied some other information like the memory requirement in each method to verify the performance relation with the execution time and memory. The initial tests of integration has developed on labtec cluster at II-UFRGS (cluster with 20 Dual Pentium III 1.1 GHz (40 nodes), 1 GB memory RAM, HD SCSI 18 GB and Gigabit Ethernet; cluster server (front-end) with Dual Pentium IV Xeon 1.8 GHz, 1 GB memory RAM, HD SCSI 36 GB and Gigabit Ethernet). We want to join the high accuracy given by C-XSC with the computational gain provided by parallelization [4].

This parallelization was developed with the tasks division among various nodes on cluster. These nodes execute the same kind of tasks and the communication between the nodes and between the nodes and the server uses message passing protocol. About the C–XSC programs executed on cluster, some changes were made in the programs for their correct use in this environment, mainly about how to manipulate *dotprecisions* variables (high accuracy variables of C–XSC) [5].

3. Solvers for Linear Systems

The algorithms implemented in our solvers were described in [6] and can be applied to any system of linear equations which can be stored in the floating point system on the computer. They will, in general, succeed in finding and enclosing a solution or, if they do not succeed, will tell the user so. In the latter case, the user will know that the problem is very ill conditioned or that the matrix A is singular. In the implementation in C–XSC, there is a chance that if the input data contains large numbers or if the inverse of A or the solution itself contain large numbers, an overflow may occur, in which case the algorithms may crash. In practical applications, this has never been observed, however. This could also be avoided by including the floating point exception handling which C–XSC offers for IEEE floating point arithmetic [7].

For this work we implemented interval algorithms for solution of linear systems of equations with dense and sparse matrices [5,8,9]. There are numerous methods and algorithms computing approximations to the solution x in floating-point arithmetic. However, usually it is not clear how good these approximations are, or if there exists a unique solution at all. In general, it is not possible to answer these questions with mathematical rigour if only floating-point approximations are used. These problems become especially difficult if the matrix A is ill conditioned. We present some algorithms which answer the questions about existence and accuracy automatically once their execution is completed successfully. Even very ill conditioned problems can be solved with these algorithms. Most of the algorithms presented here can be found in [10].

3.1. Solver for Dense Linear Systems

The C-XSC programs implemented in our solver for dense linear systems were written for the case of *real* input data (i.e. A is of type *rmatrix* and b is of type *rvector*) and for the case of the data types *interval, complex*, and *complex interval*. The changes made for the use of these other types are mainly changes of the data type of certain variables and functions in the program. Our C–XSC program *verifies the existence* of a solution and *computes an enclosure* for each of the following

types of problems:

(s) compute an enclosure for the solution of system (1) for a *square* $n \times n$ matrix A.

(o) compute an enclosure for the solution of system (1) in the *over-determined* case, i.e. for an $m \times n$ matrix A where $m > n$.

(u) compute an enclosure for the solution of system (1) in the *under-determined* case, i.e. for an $m \times n$ matrix A where $m < n$.

(S) compute an enclosure of the *inverse* A^{-1} of A.

(O) compute an enclosure of the *pseudo inverse* A^{+} of A in the *over-determined* case, i.e. for an $m \times n$ matrix A where $m > n$.

(U) compute an enclosure of the *pseudo inverse* A^{+} of A in the *under-determined* case, i.e. for an $m \times n$ matrix A where $m < n$.

This solver has two modules: the module `lss_aprx` contains the function `MINV` which computes an approximate inverse of the input matrix A of type *rmatrix* using the Gauss-Jordan algorithm (see i.e. [11]), when A is a square matrix. In the over- or under-determined case we use the Moore-Penrose pseudo inverse A^{+} of A (if A has full rank). The second module `lss` contains the functions which solve the dense linear system. This system may be square and non square ($m \times n$). In the over-determined case ($m > n$) a vector $x \in \mathbb{R}^n$ is sought whose residuum $b - Ax$ has minimal Euclidian norm whereas in the under-determined case ($n < m$) a solution $x \in \mathbb{R}^n$ is sought which has minimal norm.

3.2. Solver for Sparse Linear Systems

For the solution of a sparse linear system we present an implementation of an algorithm to compute efficiently componentwise good enclosures. Our implementation works with point as well as *interval* data (data afflicted with tolerances). We assume linear systems whose coefficient matrix has a banded structure. In this case the well known general algorithm (using the Krawczyk operator) to solve systems with dense matrices is not efficient. Since the approximate inverse R of a banded matrix A is in general a full matrix, a lot of additional storage would be required, especially if the bandwidth of A is small compared with its dimension. So a special algorithm is used to reduce the amount of storage and runtime. This method is based on the fact that matrices with banded structure are closely related to difference equations. For the banded system, we apply a LU-decomposition without pivoting (to avoid fill in) to the coefficient matrix A and derive an interval iteration similar to the well known interval iteration used in case of dense matrices. Here, however, we do not use a full approximate inverse R, but rather the interval iteration will be performed by solving two systems with banded triangular matrices L and U. The banded triangular systems are solved with the special method for difference equations described in [6]. In case of point matrices the method is designed to give almost sharp enclosures for all components (large or small in modulus) of the solution vector. A different approach to compute an enclosure for the solution vector of a large linear systems with banded or arbitrary sparse coefficient matrix (which gives enclosures with respect to the infinity norm $||\ \ ||_\infty$ only) is described in [10].

In addition to the implementation of the solution method in C–XSC, the program includes a small demonstration part (a driver) which can be used to solve some simple systems. First the program reads the number of lower and upper bands and then one value for each of the bands, i.e. initially a

Toeplitz matrix is generated. In the next step, however, any number of elements of the matrix can be changed, such that arbitrary banded matrices can be entered. To change the element $a_{i,j}$, only i, j and the new value for this element must be entered. Changing of elements is finished by entering zeros for i and j. Next the right hand side must be entered. There are several choices of predefined solutions, such that the right hand side b will be determined from this given solution. Alternatively b can be set to a constant value in all components or all components can be entered successively. In any case, the values of the components of b may be changed again similarly as for the matrix. When no changes are done anymore, the solution algorithm starts. The banded solver is called and the solution and error statistics are printed. In this way it is quite easy to explore the our C–XSC solver.

4. Basic Tests and Results

Measures and tests were made to compare the routines execution time in C/C++ language, C/C++ using MPI library, C/C++ using C–XSC library and C/C++ using C–XSC and MPI libraries. Our first tests were made to compare the routines execution time, the accuracy of results and to validate the environment developed.

4.1. Scalar product

We done tests comparing the accuracy and execution time between programs in C/C++ language (scalar product) and C/C++ using C–XSC library (optimal scalar product - with high accuracy). In the tables 1, 2 and 3 we showed the results of test with scalar product between two vectors a and b. The values of this vectors are: $a = [10^{50}, 1.25, 10^{50}, 1.1, \ldots, 10^{50}, 1.25, 10^{50}, 1.1]$ and $b = [1, 1, -1, -1, \ldots, 1, 1, -1, -1]$ with size of this vectors (n) equal to $30000, 90000$ and 180000.

Table 1
Parallel Scalar Product $a \times b$ (using 4 nodes of cluster labtec – time in seconds)

Order (n)	Program in C/C++	Program in C–XSC
30000	0.031205	0.041015
90000	0.096727	0.113847
180000	0.188670	0.225083

Table 2
Results of Scalar Product $a \times b$ in C/C++

Order (n)	Results in C/C++	Exact Results
30000	$-3.300000000000000266453525910037569701671 6$	1125
90000	$-3.300000000000000266453525910037569701671 6$	3375
180000	$-3.300000000000000266453525910037569701671 6$	6750

Table 3
Results of Scalar Product $a \times b$ in C–XSC

Order (n)	Results in C–XSC	Exact Results
30000	1124.99999999999931787897367030382156372070 3	1125
90000	3374.99999999999818101059645414352416992187 5	3375
180000	6749.99999999999636202119290828704833984375 0	6750

4.2. Matrix multiplications

We done tests with sequential and parallel programs in C/C++ language, C/C++ using MPI library, C/C++ using C–XSC library and C/C++ using C–XSC and MPI libraries. The results about the performance are showed in the table 4.

Table 4
Parallel Matrix Multiplication (using 8 nodes of cluster labtec – time in seconds)

Program/Order	512×512	1024×1024	2048×2048
C/C++ with MPI	4.603	34.131	265.284
C–XSC with MPI	115.523	916.233	7329.415

4.3. Solvers to linear systems with dense and banded matrices

To validate our solvers on clusters we will describe some tests about the quality of the results of ours solvers. How a first example, a very well known set of ill conditioned test matrices for linear system solvers are the $n \times n$ Hilbert matrices H_n with entries $(H_n)_{i,j} := \frac{1}{i+j-1}$. As a test problem, we report the results of our program for the linear systems $H_n x = e_1$, where e_1 is the first canonical unit vector. Thus the solution x is the first column of the inverse H_n^{-1} of the Hilbert matrix H_n. Since the elements of these matrices are rational numbers which can not be stored exactly in floating point, we do not solve the given problems directly but rather we multiply the system by the least common multiple lcm_n of all denominators in H_n. Then the matrices will have integer entries which makes the problem exactly storable in IEEE floating point arithmetic. For the system $(lcm_{10} H_{10})x = (lcm_{10} e_1)$, the program computes the enclosures (here an obvious short notation for intervals is used) showed in (2), which is an extremely accurate enclosure for the exact solution (the exact solution components are the integers within the computed intervals).

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} 1.000000000000000E+002, & 1.000000000000000E+002 \\ -\ 4.950000000000000E+003, & -\ 4.950000000000000E+003 \\ 7.920000000000000E+004, & 7.920000000000000E+004 \\ -\ 6.006000000000000E+005, & -\ 6.006000000000000E+005 \\ 2.522520000000000E+006, & 2.522520000000000E+006 \\ -\ 6.306300000000000E+006, & -\ 6.306300000000000E+006 \\ 9.609600000000000E+006, & 9.609600000000000E+006 \\ -\ 8.751600000000000E+006, & -\ 8.751600000000000E+006 \\ 4.375800000000000E+006, & 4.375800000000000E+006 \\ -\ 9.237800000000000E+005, & -\ 9.237800000000000E+005 \end{pmatrix} \tag{2}$$

As an other example, we compute an enclosure for a very large system. We take a symmetric Toeplitz matrix with five bands having the values $1, 2, 4, 2, 1$ and on the right hand side we set all components of b equal to 1. Then the program produces the following output for a system of size $n = 200000$ (only the first ten and last ten solution components are printed):

```
Dimension  n = 200000
Bandwidths l,k : 2 2
A = 1 2 4 2 1
change elements ? (y/n) n
```

```
b = =1
change elements ? (y/n) n

x =
  1: [  1.860146067479180E-001,  1.860146067479181E-001 ]
  2: [  9.037859550210300E-002,  9.037859550210302E-002 ]
  3: [  7.518438200412189E-002,  7.518438200412191E-002 ]
  4: [  1.160876404875081E-001,  1.160876404875082E-001 ]
  5: [  1.003153932563721E-001,  1.003153932563722E-001 ]
  6: [  9.427129202687645E-002,  9.427129202687647E-002 ]
  7: [  1.028361799416204E-001,  1.028361799416205E-001 ]
  8: [  1.005240450090008E-001,  1.005240450090009E-001 ]
  9: [  9.874921290539136E-002,  9.874921290539138E-002 ]
 10: [  1.004617422430963E-001,  1.004617422430964E-001 ]

199990: [  1.001953939326196E-001,  1.001953939326197E-001 ]
199991: [  1.004617422430963E-001,  1.004617422430964E-001 ]
199992: [  9.874921290539136E-002,  9.874921290539138E-002 ]
199993: [  1.005240450090008E-001,  1.005240450090009E-001 ]
199994: [  1.028361799416204E-001,  1.028361799416205E-001 ]
199995: [  9.427129202687645E-002,  9.427129202687647E-002 ]
199996: [  1.003153932563721E-001,  1.003153932563722E-001 ]
199997: [  1.160876404875081E-001,  1.160876404875082E-001 ]
199998: [  7.518438200412189E-002,  7.518438200412191E-002 ]
199999: [  9.037859550210300E-002,  9.037859550210302E-002 ]
200000: [  1.860146067479180E-001,  1.860146067479181E-001 ]

max. rel. error =  1.845833860422451E-016  at i = 3
max. abs. error =  2.775557561562891E-017  at i = 1
min. abs. x[3] = [7.518438200412189E-002, 7.518438200412191E-002]
max. abs. x[1] = [1.860146067479180E-001, 1.860146067479181E-001]
```

Our last example is about the matrix inversion (test about accuracy).

In this example $A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon^2 \end{pmatrix}$ and $A^{-1} = \begin{pmatrix} 1 & -\frac{1}{\epsilon} & 0 \\ 0 & \frac{1}{\epsilon} & 0 \\ 0 & 0 & \frac{1}{\epsilon^2} \end{pmatrix}$.

With $\epsilon = 1.084202172485504E-019$, our program in C–XSC obtained:

$$A = \begin{pmatrix} 1.0E+000, & 1.000000000000000E+000, & 0.000000000000000E+000 \\ 0.0E+000, & 1.084202172485504E-019, & 0.000000000000000E+000 \\ 0.0E+000, & 0.000000000000000E+000, & 1.175494350822288E-038 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 1.0E+000, & 1.000000000000000E+000, & 0.000000000000000E+000 \\ 0.0E+000, & 7.136238463529799E+044, & 0.000000000000000E+000 \\ 0.0E+000, & 0.000000000000000E+000, & 8.507059173023462E+037 \end{pmatrix}.$$

5. Solving Real Life Applications With High Accuracy

In the original solvers we included three new methods to solve linear systems: Conjugate Gradient, Householder and Givens methods. All these methods have versions with and without the high accuracy characteristic. We use this methods in real life applications like hydrodynamic (parallel computational model with local refinement and dynamic load balancing for the simulation of substances transportation and hydrodynamic – [12]), agriculture (optimization of the air distribution in grain storehouse with aeration of the mass of grains – [13]) and power electric systems [14]. This methods and applications are research topics of ours research groups in Brazil [9].

In our research we are implementing other methods to solve linear systems. We are implementing, initially, the sequential versions of Gauss-Seidel, Gauss-Jacobi, Gauss Elimination and LU Decomposition methods [15–17]. Nowadays, we are implementing the parallel versions of these methods with and without high accuracy (these programs are been implemented only with the high accuracy characteristic, we are not using interval arithmetic).

6. Conclusions

In our research some programs were developed in C–XSC with the validated numeric paradigm, where the results are obtained with a good quality. The main contributions of our work are:

- integration between the libraries C–XSC and MPI;
- effective use of library C-XSC on cluster computers;
- resolution of linear systems with high accuracy.

In our work we provide the development of selfverifying solvers for linear systems of equations with dense and sparse matrices and the integration between C–XSC and MPI libraries on cluster computers. This integration was not trivial because was necessary to send correctly the special high accuracy variables (*dotprecision*) of C–XSC to the cluster processors using the library MPI without lost of the high accuracy characteristic. Our software run on clusters at UFRGS, UPF and Wuppertal and the integration between C-XSC and MPI was done correctly. Our tests with matrix multiplication, scalar product and methods to solve linear system of equations show that the C-XSC library needs to be optimized to be efficient in a High Performance Environment. This optimization is other research topic of Brazilian groups.

Nowadays we are working in the implementation of parallel versions of methods to solve linear systems (without and with high accuracy). These methods are used in real life applications like hydrodynamic (parallel computational model with local refinement and dynamic load balancing for the simulation of substances transportation and hydrodynamic), agriculture (optimization of the air distribution in grain storehouse with aeration of the mass of grains) and power electric systems.

Acknowledgement

This work is supported in part by CAPES and FAPERGS (Brazil) and DAAD (Germany). It is part of a international cooperation project between Brazilian universities (Universidade Federal do Rio Grande do Sul, Pontifícia Universidade Católica do Rio Grande do Sul and Universidade de Passo Fundo) and German universities (Universität Wuppertal and Universität Karlsruhe).

References

[1] Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *C-XSC Toolbox for Verified Computing I: basic numerical problems*. Springer-Verlag, Berlin/Heidelberg/New York, 1995.

[2] Hofschuster, W., Krämer, W., Wedner, S., Wiethoff, A.: *C-XSC 2.0: A C++ Class Library for Extended Scientific Computing*. Universität Wuppertal, Preprint BUGHW - WRSWT 2001/1 (2001).

[3] Hölbig, C.A., Krämer, W.: *Selfverifing Solvers for Dense Systems of Linear Equations Realized in C-XSC*. Universität Wuppertal, Preprint BUGHW - WRSWT 2003/1, Wuppertal, 2003.

[4] Hölbig, C.A., Diverio, T.A., Claudio, D.M., Krämer, W., Bohlender, G.: Automatic Result Verification in the Environment of High Performance Computing In: IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, 2002, Paris. Extended abstracts, pg. 54-55 (2002).

[5] Hölbig, C.A., Morandi Júnior, P.S., Alcalde, B.F.K., Diverio, T.A., Claudio, D.M.: Solving Linear Systems on Cluster Computers with High Accuracy. In: VII WORKSHOP ON STATE-OF-THE-ART IN SCIENTIFIC COMPUTING, 2004, Copenhagen. Book of Abstracts, pg. 28–29. Copenhagen, 2004.

[6] Krämer, W., Kulisch, U., Lohner, R.: *Numerical Toolbox for Verified Computing II - Advanced Numerical Problems*. University of Karlsruhe (1994), see *http://www.uni-karlsruhe.de/~Rudolf.Lohner/papers/tb2.ps.gz*.

[7] American National Standards Institute / Institute of Electrical and Electronics Engineers: *A Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std. 754-1985, New York, 1985.

[8] Hölbig, C.A., Krämer, W., Diverio, T.A.: *An Accurate and Efficient Selfverifying Solver for Systems with Banded Coefficient Matrix*. In [19], pp. 283–290, 2004.

[9] Hölbig, C.A., Kolberg, M.L., Morandi Júnior, P.S., Alcalde, B.F.K., Diverio, T.A., Claudio, D.M.: Solvers with High Accuracy to Linear Systems on Clusters. In: XI INTERNATIONAL CONGRESS ON COMPUTATIONAL AND APPLIED MATHEMATICS, 2004, Leuven. Abstracts of Talks, pg. 70. Leuven, 2004.

[10] Rump, S. M.: *Validated Solution of Large Linear Systems*. In [18], pp 191–212, 1993.

[11] Stoer, J., Bulirsch, R.: *Introduction to Numerical Analysis*. Springer-Verlag, New York, 1980.

[12] Rizzi, R.L., Dorneles, R.V., Piccinin Júnior, D., Martinotto, A.L., Hölbig, C.A., Navaux, P.O.A., Diverio, T.A.: *Parallelization of Krylovs Subspace Methods in Multiprocessor PC Cluster*. In [19], pp. 543–550, 2004.

[13] Khatchatourian, O., Savicki, L.D.: Optimization of the air distribution in Grain Storehouse with Aeration in No-Uniform Conditions of the Mass of Grains. In: 16th Brazilian Congress of Mechanical Engineering, 2001, Uberlândia. Proceedings, pp. 73–82, Uberlândia, 2001.

[14] Barboza, L.V., Dimuro, G.P., Reiser, R.H.S.: Towards Interval Analysis of the Load Uncertainty in Power Electric Systems. In: 8th International Conference on Probability Methods Applied to Power Systems, 2004, Washington. Proceedings, pp 1–6, Washington, 2004.

[15] Dongarra, J., Foster, I., Fox, G., Gropp, W., Kennedy, K., Torczon, L., White, A.: *Sourcebook of Parallel Computing*. Morgan Kaufmann Publishers, San Francisco. 2003.

[16] Karniadakis, G.E., Kirby II, R.M.: *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation*. Cambridge University Press, Cambridge. 2003.

[17] Yousef Saad: *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia. 2003.

[18] Albrecht, R., Alefeld, G., Stetter, H. J. (Eds.): *Validation Numerics – Theory and Applications*. Computing Supplementum **9**, Springer-Verlag (1993).

[19] Joubert, G.R., Nagel, W.E., Peters, F.J., Walter, W.V. (Org.): Parallel Computing: Software Technology, Algorithms, Architectures, and Applications. Elsevier Science Publishers, Londres, 2004.